Compilation et programmation *constant-time*

Keywords Compilation, Sécurité, Preuve de transformation de programme, langage C
Main advisor David Pichardie (`David.Pichardie@irisa.fr`)
Second advisor Sandrine Blazy (`Sandrine.Blazy@irisa.fr`)
Localisation Équipe Celtique IRISA/Inria, Rennes

Certains programmes manipulent des données sensibles (des clés secrètes dans les implémentations des primitives cryptographiques par exemple) qu'il convient de protéger d'un attaquant observant les calculs. Une attaque par canaux cachés particulièrement redoutable consiste à observer les différences de temps de calcul entre plusieurs exécutions pour apprendre des informations sensibles. De telles attaques ont été montées sur des implémentations de RSA [3] ou AES [5].

La discipline de programmation *constant-time* est une contre-mesure largement répandue pour éviter ces failles. Elle s'interdit d'utiliser des tests, des accès mémoires, voire des multiplications sur des données teintées par le secret. Des travaux récents proposent des techniques de vérification automatiques pour assurer qu'un programme adhère bien à une telle discipline [1, 2]. Ces techniques utilisent des analyses statiques avancées qui opèrent sur une représentation à haut niveau des programmes. Il convient alors de s'assurer que la compilation de ces programmes vers du code exécutable ne va pas altérer la propriété de sécurité.

En effet, certaines optimisations peuvent invalider la propriété *constant-time* d'un programme [4, 8]. L'objectif de ce stage est d'étudier les compilateurs LLVM [6] et CompCert [7] du langage C pour comprendre comment les transformations qu'ils mettent en œuvre interagissent avec la propriété *constant-time*. En fonction des résultats de cette étude, l'étudiant.e pourra soit proposer une modification de la transformation incriminée pour lui faire respecter la manipulation des valeurs teintées, soit proposer une preuve de correction (sans assistant de preuve) pour les transformations les plus intéressantes. Nous recherchons un.e stagiaire motivé.e par la manipulation symbolique de programme (traduction, optimisation, analyse statique, preuve) et à l'aise en programmation OCaml et C++. Le stage devra faire l'objet d'un premier prototype dans le compilateur CompCert et d'une petite formalisation sémantique sans assistant de preuve.

Références

[1] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. Verifying constant-time implementations. In *25th USENIX Security Symposium, USENIX Security 16*, 2016.

[2] Sandrine Blazy, David Pichardie, and Alix Trieu. Verifying constant-time implementations by abstract interpretation. In *22nd European Symposium on Research in Computer Security, ESORICS'17*, 2017.

[3] David Brumley and Dan Boneh. Remote timing at- tacks are practical. In *Comp. Networks, 48(5)*, 2005.

[4] Bart Coppens, Ingrid Verbauwhede, Koen De Bosschere, and Bjorn De Sutter. Practical mitigations for timing-based side-channel attacks on modern x86 processors. In *Security and Privacy*, pages 45–60, 2009.

[5] David Gullasch, Endre Bangerter, and Stephan Krenn. Cache games - bringing access-based cache attacks on aes to practice. In *32nd IEEE Symposium on Security and Privacy*, 2011.

[6] Chris Lattner and Vikram S. Adve. LLVM : A compilation framework for lifelong program analysis & transformation. In *CGO*, 2004.

[7] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7), 2009.

[8] David Molnar, Matt Piotrowski, David Schultz, and David Wagner. The program counter security model : Automatic detection and removal of control-flow side channel attacks. In *Info. Security and Crypto.*, 2006.